Segunda Prova Prática de Estrutura de Dados

📌 Instruções Gerais:

- A prova é individual.
- É permitido consultar seu próprio material e anotações.
- Linguagem: **C++**.

(4 pontos) Exercício 1: Verificação de Conectividade em Grafo com DFS

Objetivo: Implementar o algoritmo de **Busca em Profundidade (DFS)** para verificar se um grafo é **conectado**.

Instruções (Dicas):

1. Crie um grafo **não direcionado** representado por **lista de adjacência**.
2. O grafo deve ter pelo menos **7 vértices e 9 arestas**.
3. Implemente a função `bool ehConectado(Grafo* grafo)` que verifica se **todos os vértices estão conectados** (i.e., se existe um caminho entre qualquer par de vértices).
4. No `main`, monte ao menos dois grafos: um conectado e um **não conectado**, e imprima os resultados.

Exemplo de saída esperada:

Grafo 1: Conectado? Sim
Grafo 2: Conectado? Não

> 💡 Dica: use DFS ou BFS como base, mas foque em visitar todos os vértices a partir de um único ponto.

(3 pontos) Exercício 2: Ordenação de Cores com Ordem Personalizada (20 Cores)

Objetivo: Implementar um algoritmo de ordenação para organizar uma lista de cores com base em uma **ordem definida manualmente, não alfabética**.

Ordem Predefinida de Cores:

Use a seguinte ordem como referência (você pode imprimi-la no início do programa para facilitar):

1. preto

2. marrom

3. vermelho

4. laranja

5. amarelo

6. verde

7. azul

8. anil

9. violeta

10. rosa

11. cinza

12. branco

13. dourado

14. prateado

15. ciano

16. magenta

17. lilás

18. vinho

19. salmão

20. turquesa

Instruções (Dicas):

1. O usuário deve digitar **15 nomes de cores**, escolhidos **entre as cores da ordem acima** (podem se repetir).
2. Implemente um algoritmo de ordenação (a sua escolha) que utilize essa ordem fixa e **não** a ordem alfabética.
3. Use uma função auxiliar como:

   int indiceCor(string cor);

   que retorna a **posição da cor** na lista acima (por exemplo, `"vermelho"` retorna 2, `"turquesa"` retorna 19).

4. A ordenação deve comparar os nomes das cores com base nessa posição.

Exemplo de Entrada:

Entrada do usuário (15 cores aleatórias):

vinho, azul, rosa, vermelho, dourado, cinza, azul, amarelo, lilás, rosa, vermelho, anil, verde, branco, preto

Saída Esperada:

Lista original:

vinho, azul, rosa, vermelho, dourado, cinza, azul, amarelo, lilás, rosa, vermelho, anil, verde, branco, preto

Lista ordenada:

preto, vermelho, amarelo, verde, azul, azul, anil, rosa, rosa, cinza, branco, dourado, lilás, vinho, vermelho

> 💡 Dica: se quiser facilitar e souber usar, use `vector<string>` em vez de array tradicional, e compare usando a função `indiceCor`.

(3 pontos) Exercício 3: Implementação de Tabela Hash com Tratamento de Colisões por Encadeamento

Objetivo: Implementar uma tabela hash simples em C++ utilizando uma **função hash personalizada e diferenciada**.

Instruções:

1. Crie uma tabela hash de tamanho **17**.
2. Você deverá implementar uma função hash personalizada com a seguinte lógica:
    a. Considere os valores ASCII de cada caractere do nome.
    b. Para cada caractere na posição i, multiplique seu valor ASCII por P^i, onde:
    c. P é um número primo fixo, por exemplo 7
    d. i é a posição do caractere (começando do 0)
    e. Some todos esses valores.
    f. Por fim, aplique o módulo 17 para obter o índice da tabela.
3. Implemente as funções:
    a. `void inserir(string tabela[], string nome)`
        i. Se a posição já estiver ocupada, insira e exiba: `"Colisão ao inserir <nome> na posição <indice>.`
    b. `void imprimirTabela(string tabela[])`

4. No `main`, insira pelo menos **10 nomes** (podem causar colisão), e depois imprima a tabela final.

Exemplo com **`"Leo"`** e **`P = 7`**:

- `'L'` (76) × 7^0 = 76 × 1 = **76**
- `'e'` (101) × 7^1 = 101 × 7 = **707**
- `'o'` (111) × 7^2 = 111 × 49 = **5439**

Soma total: 76 + 707 + 5439 = 6222
Hash final: 6222 % 17 = 0
Resultado: `"Leo"` deve ser inserido na posição 0.

Exemplo de Saída:

```
Tentando inserir: "Alice" → índice 2

Tentando inserir: "Bruno" → índice 2

Colisão ao inserir Bruno na posição 2.

Tentando inserir: "Carlos" → índice 7

Tentando inserir: "Diego" → índice 0

Tentando inserir: "Elisa" → índice 14

...

Tabela Hash Final:

0: Diego

2: Alice - Bruno

7: Carlos

14: Elisa
```

💡 Dica: Use `#include <cmath>` para facilitar a contar, use pow().